Practice Midterm 2 Solutions

This page contains solutions to Practice Midterm 2.

Q1) C++ fundamentals

Below we give two different solutions, there are other alternate approaches that also work.

```
void trimPrefix(string& str) {
    // must order tests to confirm index is valid before accessing it
    while (str.size() > 0 && !isalpha(str[0])) {
        str.erase(0, 1); // modify str in-place
    }
}

void trimPrefixAlt(string& str) {
    int i;
    for (i = 0; i < str.size(); i++) {
        if (isalpha(str[i])) {
            break;
        }
    }
    str = str.substr(i); // overwrite str with shortened string
}
```

```
STUDENT_TEST("My test cases for trimPrefix") {
    string input = "#3.14&@!"; // try input containing NO alpha
    trimPrefix(input);
    EXPECT_EQUAL(input, "");

    input = "abcABC";      // try input containing ONLY alpha
    trimPrefix(input);
    EXPECT_EQUAL(input, "abcABC");
}
```

Q2) ADTs

```
Map<string, string> collect(Set<string> huntList, Map<string, Set<string>>&
campus) {
    Map<string, string> result;

    for (string place: campus) { // iterate over places on campus
        Set<string> foundHere = campus[place]; // items at place
        foundHere.intersect(huntList);  // winnow to only items on list
        if (foundHere.size() >= 2) { // place has at least 2 items needed
            huntList.difference(foundHere); // cross items off list
            for (string item: foundHere) {
                result[item] = place;   // assign item to result map
            }
        }
        if (huntList.isEmpty()) break;  // done if have collected all
    }
    return result;
}
```

Q3) Code study: ADTs and Big-O

cleaveVector

```
a = {6, 1, 5}
b = {3, 9}
```

cleaveSet

```
a = {5, 6, 9} // elements listed in any order form same set
b = {1, 3}
```

cleaveStack

```
a = {9, 3, 6}
b = {5, 1}
```

cleaveQueue

```
a = {6, 1, 5}
b = {9, 3}
```

	k=2	k=a.size()/2
cleaveVector	O(N)	$O(N^2)$
cleaveSet	O(logN)	O(NlogN)
cleaveStack	O(1)	O(N)
cleaveQueue	O(1)	O(N)

Q4) Recursion

```
int countHelper(int n, int k, int nInRow) {
    if (nInRow > k) {   // too many repeats, this sequence not kGood
        return 0;
    }
    if (n == 0) {   // this sequence complete and is kGood
        return 1;
    }
    return countHelper(n-1, k, nInRow + 1) + countHelper(n-1, k, 1);
}

int countKGood(int n, int k) {
    return countHelper(n, k, 0);
}
```

Q5) Recursive backtracking

```
int maxHelper(Vector<string>& symbols, Lexicon& lex, string soFar) {
    int best = 0; // track best of this call/all recursive calls

    if (!lex.containsPrefix(soFar)) {
        return 0;   // this is dead end, prune
    }
    if (lex.contains(soFar)) {
        best = soFar.length();  // found a word!
    }
    for (int i = 0; i < symbols.size(); i++) {
        string choice = symbols[i];
        symbols.remove(i);  // no repeats => remove choice from vector
        best = max(best, maxHelper(symbols, lex, soFar + choice)); // update
best
        symbols.insert(i, choice); // unchoose => restore to vector
    }
    return best;
}

int maxLength(Vector<string>& symbols, Lexicon& lex) {
    return maxHelper(symbols, lex, "");
}
```

Just for fun, the longest such word is "IrReCoNCILaBiLiTiEs" (length 19).